

Release Notes

XMLMill for Java - XMLMill for Domino

3.00

xmlmill

© 2000-2008 Pecunia Data Systems bvba. All rights reserved.

NOTICE: All information contained herein is the property of Pecunia Data Systems bvba.

This publication and the information herein are furnished AS IS, are subject to change without notice, and should not be construed as a commitment by Pecunia Data Systems bvba. Pecunia Data Systems bvba assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third-party rights.

Table of Contents

1. Preface	1
2. What's new ?	1
2.1. XMLMill for Java.	1
2.2. XMLMill for Domino.	1
3. Upgrading from previous versions	1
3.1. Upgrading from version 2.80 (or later).	1
3.1.1. Installation.	1
3.1.1.1. Unzip the XMLMill Distribution.	1
3.1.1.2. Update your classpath / Domino Environment.	1
3.1.2. New Logging implementation.	1
3.1.3. New elements in the config.xml configuration file.	1
3.1.4. New xmlmill.xsd Schema.	2
3.1.5. New DPI setting for images.	2
3.2. Upgrading from earlier versions.	2
4. XMLMill for Java	1
4.1. Complete Acrofields of an existing PDF document.	1
4.1.1. Make a new or modify an existing PDF document.	1
4.1.2. Define an xml structure.	1
4.1.3. Define an xsl document.	2
4.1.3.1. Define the location of the 'template'.....	2
4.1.3.1.1. Attribute: template.	2
4.1.3.1.2. Attribute: template-in-memory.	2
4.1.3.2. Define the content of the acrofields.....	2
4.1.3.2.1. Element: ml:acroform.	2
4.1.3.2.2. Element: ml:acrofield.	3
4.1.3.2.3. Generate the document.	3
4.2. Stamp the page(s) of an existing PDF document.	3
4.2.1. Refer to an existing PDF document.	3
4.2.1.1. Define the location of the 'template'.....	4
4.2.1.1.1. Attribute: template.	4
4.2.1.1.2. Attribute: template-in-memory.	4
4.3. Complete Acrofields and Stamp the page(s) of an existing PDF document.	4
4.4. XMLMill InterActive: New user-interface.	4
4.4.1. Splash screen.	5
4.4.2. Main Window.	5
4.4.2.1. Dashboard.	5
4.4.2.2. Logging.	6
4.5. XMLMill Logging.	6
4.5.1. Class-path changes.	6
4.5.2. Changing .java code.	7
4.5.2.1. Example: Using the com.xmlmill.log.FileLogger.	7
4.5.2.2. Example: Using the log4j package: org.apache.log4j.	8
4.5.2.3. Example: Using the JDK 1.4 package: java.util.logging.	9
4.6. XMLMill Configuration.	10
4.6.1. Introduction.	10
4.6.2. Element: <legacy>.	10
4.6.2.1. Element: <external-graphic>.	11
4.6.3. Element: <templates>.	11

4.6.3.1. Element: <template>.....	11
4.7. Implementation: break-before = 'column' attribute/value.....	11
4.8. Implementation: break-after = 'column' attribute/value.....	11
4.9. Rotate an image in a table-cell.....	12
4.10. Behavioral change: Empty ml:textbox or ml:table-cell.....	12
4.10.1. Before version 2.80.....	12
4.10.2. As of version 2.80.....	12
4.11. Behavioral change: The calculation of the width or height of an image has ..	12
changed.....	13
4.12. XMLMill Batch.....	13
4.13. Removed classes.....	13
4.14. Renamed classes.....	14
5. XMLMill for Domino.....	1
5.1. Complete Acrofields of an existing PDF document.....	1
5.2. Stamp the page(s) of an existing PDF document.....	1
5.3. Complete Acrofields and Stamp the page(s) of an existing PDF document. . .	1
5.4. XMLMill Configuration.....	1
5.5. Implementation: break-before = 'column' attribute/value.....	1
5.6. Implementation: break-after = 'column' attribute/value.....	1
5.7. Rotate an image in a table-cell.....	1
5.8. Behavioral change: Empty ml:textbox or ml:table-cell.....	1
5.9. Behavioral change: The calculation of the width or height of an image has ...	1
changed.....	1
6. Bugfixes.....	1
7. Appendix: Adding Acrofields to a PDF document with Acrobat ..	1
Professional 8.....	1
7.1. Modify an existing document with Acrobat Professional (8.0).....	1

1. Preface

The XMLMill application is intended for software developers who want to generate .pdf documents from xml and/or xsl.

For an overview of how to use XMLMill in general please consult in the **docs/** directory:.

- ◆ **apidoc/** -- The JavaDoc concerning the **XMLMill** public api.
- ◆ **userguide.pdf** -- XMLMill user's guide.
- ◆ **dtdguide.pdf** -- An explanation of XMLMill's elements (tags) and their attributes.
- ◆ **samplesguide.pdf** -- An overview of the examples in the **samples/** directory contained in the download.
- ◆ **digitalsignatures.pdf** -- An overview of how to digitally sign PDF documents (one pass generation and signing).

If you have questions, please do not hesitate to send a mail to support@xmlmill.com.

☞ *This document is completely generated with XMLMill 3.00 using **rnotes300.xml** and **rnotes.xsl**. These files can be found in the directory **samples/docs/rnotes** in the download.*

xmlmill

2. What's new ?

2.1. XMLMill for Java

This release notes document describes all new (or enhanced) functionalities of this release. The main changes are:

- ◆ Complete the Acrofields of an existing PDF document (acting as 'template') and write the result into a new document.
- ◆ Stamp the page(s) of any existing PDF document using the regular **ml:** elements (this means adding new data on the document page(s)) and write the result into a new document.
- ◆ Complete the Acrofields and stamp the page(s) of an existing PDF document (acting as template) and write the result into a new document.
- ◆ XMLMill InterActive has a new user-interface, enhancing greatly the user-experience.
- ◆ The logging mechanism has been changed to include Apache's commons-logging-1.1 functionality instead of Log4j (although Log4j can still be used).
- ◆ The DPI (dots per inch) of each image is now read from the image file itself (if applicable) instead of a fixed 96 DPI, leading to smaller (or bigger) image representation in the PDF document compared with the image itself.
- ◆ An entry has been added in the **config.xml** file regarding the use of the DPI settings for backwards compatibility (facilitating the upgrade to this version).
- ◆ An entry has been added in the **config.xml** file regarding preloading 'template' documents so completing Acrofields or stamping existing documents is extremely fast (and minimizing memory usage).
- ◆ Implementation of the **break-before = `column'** attribute/value pair.
- ◆ Implementation of the **break-after = `column'** attribute/value pair.
- ◆ An image can be rotated in a **table-cell**.
- ◆ Behavioral change: The calculation of the width or height has been changed, eliminating a wrong calculation if the width or height of the image was larger than the parent's content-area.
- ◆ Behavioral change: An empty **ml:textbox** or **ml:table-cell** will no longer have an implicit space character added to it.
- ◆ The loglevels **ALL** and **OFF** when using XMLMill in Batch were suppressed.
- ◆ Removal of deprecated classes: **PDXLogger**, **PDXLogFile**, **PDELogFile**, **PDXTransform**.
- ◆ Following class has been renamed: **PDXTransformErrorHandler** has become **JAXPErrrorHandler**.
- ◆ Bug fixes.

2.2. XMLMill for Domino

This release notes document describes all new (or enhanced) functionalities of this release. The main changes are:

- ◆ Complete the Acrofields of an existing PDF document (acting as 'template') and write the result into a new document.
- ◆ Stamp the page(s) of any existing PDF document using the regular **ml:** elements (this means adding new data on the document page(s)) and write the result into a new document.
- ◆ Complete the Acrofields and stamp the page(s) of an existing PDF document (acting as template) and write the result into a new document.

- ◆ XMLMill InterActive has a new user-interface, enhancing greatly the user-experience.
- ◆ The DPI (dots per inch) of each image is now read from the image file itself (if applicable) instead of a fixed 96 DPI, leading to smaller (or bigger) image representation in the PDF document compared with the image itself.
- ◆ An entry has been added in the **config.xml** file regarding the use of the DPI settings for backwards compatibility (facilitating the upgrade to this version).
- ◆ An entry has been added in the **config.xml** file regarding preloading 'template' documents so completing Acrofields or stamping existing documents is extremely fast (and minimizing memory usage).
- ◆ Implementation of the **break-before = `column`** attribute/value pair.
- ◆ Implementation of the **break-after = `column`** attribute/value pair.
- ◆ An image can be rotated in a **table-cell**.
- ◆ Behavioral change: The calculation of the width or height has been changed, eliminating a wrong calculation if the width or height of the image was larger than the parent's content-area.
- ◆ Behavioral change: an empty **ml:textbox** or **ml:table-cell** will no longer have an implicit space character added to it.
- ◆ The loglevels **ALL** and **OFF** when using XMLMill in Batch were suppressed.
- ◆ The **template_rt_html.xsl** (for converting rich-text to **ml:** elements) has been modified to make it more compatible with the Tine MCE Editor.
- ◆ Bug fixes.

xmlmill

3. Upgrading from previous versions

3.1. Upgrading from version 2.80 (or later)

Following changes need to be addressed to your environment before you can use this version:

VERY IMPORTANT

☞ *PLEASE BACKUP YOUR DATA BEFORE STARTING THE UPGRADE.*

3.1.1. Installation

3.1.1.1. Unzip the XMLMill Distribution

1. Unzip (untar) the downloaded distribution in a directory (for example: `/xmlmill`).

☞ *The **lib/** directory includes by default the JAXP 1.2 compliant Xerces parser (**xercesImpl.jar**) and the JAXP 1.2 compliant Xalan transformer (**xalan.jar**, **xml-apis.jar**). It also includes Apache's common logging package (**commons-logging-1.1.jar**).*

3.1.1.2. Update your classpath / Domino Environment

Place following packages on the classpath:

1. Replace your existing **xmlmill.jar** file with the one found in the **lib** directory.
2. Remove the **log4j-*.jar** package and replace it with the **commons-logging-1.1.jar** package as found in the **lib** folder.

☞ *Domino 6 users can find these files in the `...lotus/domino/data/domino/servlet` directory (if you have followed the **XMLMill for Domino: Installation Guide R6** when you installed XMLMill*

☞ *Domino 7 users can find these files in the `.../jvm/lib/ext` directory.*

☞ *In case you already use a previous version of **commons-logging-1.1.jar**, you can use that version.*

3.1.2. New Logging implementation

Applicable for: XMLMill for Java

Impact on: Java CodeBase.

Uptill the previous version of XMLMill the **log4j** package was used to log the information during the generation process.

As this was sometimes a constraint for using XMLMill in some environments, the current version uses Apache's commons-logging-1.1 package. This is an ultra-thin bridge between different logging implementations. This facilitates using XMLMill with any logging implementation at runtime.

☞ *Please consult the [XMLMill Logging](#) chapter in this document for more information.*

☞ *For XMLMill for Domino developers there is no impact. All changes have already been implemented in the Domino connector.*

☞ *Depending on your situation you still might need the **log4j-*.jar** package.*

3.1.3. New elements in the **config.xml** configuration file

Applicable for: XMLMill for Java; XMLMill for Domino.

Impact on: Configuration.

In case the default `config.xml` file is not used some new elements need to be copied into the existing `config.xml` file.

The new entries are:

- ◆ `<legacy>` and children elements.
- ◆ `<templates>` and children elements.

For more information regarding what to do please visit the [XMLMill Configuration](#) chapter in this document

3.1.4. New `xmlmill.xsd` Schema

Applicable for: XMLMill for Java; XMLMill for Domino.

Use the new Schema with your existing `.xsl` documents in order to use the new functionalities of this release. No other code changes are necessary in the existing `.xsl/.mill` document.

☞ You can find the latest `xmlmill.xsd` schema in the download in the `docs/xsd` directory.

3.1.5. New DPI setting for images

Applicable for: XMLMill for Java; XMLMill for Domino.

Impact on: Configuration and/or `.xsl/.mill` documents.

In previous versions the DPI (dots per inch) was always defined at 96 DPI for each image instead of using the DPI setting of the image itself (if applicable).

As of this version XMLMill will use the DPI setting as indicated in the image file itself.

☞ The only image type that has DPI settings stored in the image file itself is **JPG**. Other image types will keep the DPI of 96.

This could impact the generation of your existing documents as images (of **JPG** type only) can be generated smaller or bigger than before (as the DPI setting is no longer fixed at 96 DPI).

Two options can be done to solve this problem:

- ◆ Modify your documents or images so the new generated document has the same output as before.
- ◆ Configure XMLMill so it is backwards compatible with previous versions (by modifying the `readDPIfromImage` attribute of the `<legacy>` element in the configuration file (`config.xml`)).

For more information regarding option 2 please visit the [XMLMill Configuration](#) chapter in this document

Check the `Example_dpi-v300.mill` document as an example in the `samples/mill` directory in the download.

3.2. Upgrading from earlier versions

The best procedure to make your transition as smooth as possible is as follows:

1. Back up all your `xml/xsl/mill` files or Java code.
2. Read all release notes <http://www.xmlmill.com/news.html> from your current version to the latest version in correct order (from oldest release notes to newest).
3. Modify your `xml/xsl/mill` files or Java code accordingly to the release notes.

☞ Previous versions can be found at <http://www.xmlmill.com/download/xmlmillforjava.html#pv> or <http://www.xmlmill.com/download/xmlmillfordomino.html#pv>

4. XMLMill for Java

4.1. Complete Acrofields of an existing PDF document

As of this version you can complete Acrofields of an existing PDF document, using this PDF document as a template. This has two big advantages:

- ◆ You can design documents in a PDF editor (for example Acrobat Professional), eliminating cumbersome 'design' of the PDF document using `m1:/xsl` elements.
- ◆ The speed of generating documents can be increased considerably by using an existing PDF document as a 'template', thanks to the caching technique used by XMLMill.

☞ *A template pdf document is in fact a PDF file with an AcroForm.*

☞ *It is not possible to encrypt or digitally sign a generated PDF document based on a template. This will be solved in the next version.*

The following steps are needed in order to use this functionality:

1. Make a new or modify an existing PDF document so it includes the so-called Acrofields (check your PDF-editor for more information).
2. Define an .xml document (as you have done before - nothing new).
3. Define a .xsl document containing the **AcroForm** and **acrofield** elements (see below for more information).
4. Generate the document.

☞ *For a complete example please check the **Element_acrofield_FormFieldsEnumerator-form-only.mill** example in the **samples/mill/** directory contained in the download.*

4.1.1. Make a new or modify an existing PDF document

In case you already have an existing document with Acrofields there is a good chance you can use it without any modification.

In case you still have to make the PDF document you can do so by using one of the existing PDF-editors (check the internet for such applications).

☞ *The PDF document should preferably be of version 1.3 or 1.4 (higher version might have an internal PDF structure which XMLMill cannot read (yet)).*

☞ *The Acrofields should not be defined as fully qualified names. Although XMLMill can perfectly handle these names, the fields are shown empty in Acrobat Reader (although some other readers do show the fields correctly).*

☞ *Mark all Acrofields immediately as **read-only**.*

☞ *Please read the [Appendix: Adding Acrofields to a PDF document with Acrobat Professional 8](#) chapter for more information on how to do this in Acrobat Professional 8.*

4.1.2. Define an xml structure

Define an xml-structure that will contain the content of the fields, for example:

```
<data>
  <customerID>KL-123458</customerID>
  <invoicenum>#2007-21-01-589</invoicenum>
  ...
</data>
```

☞ Preferably use the same names for the elements and for the acrofields in the PDF document, for example:

4.1.3. Define an xsl document

In order to use a pdf document as a template following needs to be done:

1. Define the location of the PDF document acting as 'template'.
2. Define the content of the acrofields.

4.1.3.1. Define the location of the 'template'.

The `ml:document` element has been extended with 2 new attributes:

4.1.3.1.1. Attribute: `template`

Value: <name>

Initial: an empty name

Applies to: ml:document

Inherited: No, a value is required

Description:

The `template` attribute defines the location of the PDF document that will act as 'template'.

Values have the following meaning:

<name> This value indicates the name (and location) of the PDF document (acting as 'template') to use.

Example:

```
<ml:document template="file:pdftemplates/FormFieldsEnumerator-final-1.3.pdf"
template-in-memory="on" ...>
```

4.1.3.1.2. Attribute: `template-in-memory`

Value: on | off | true | false

Initial: off

Applies to: ml:document

Inherited: No, a value is required

Description:

The `template-in-memory` attribute defines whether the template, after it has been parsed, must be kept in memory so a second reading of the document can be avoided at the moment of generating the new document based on this template.

```
<ml:document template="file:pdftemplates/FormFieldsEnumerator-final-1.3.pdf"
template-in-memory="on" ...>
```

☞ This will not prevent that generating multiple documents with the same .xsl document the 'template' PDF document will each time be parsed again!

☞ To increase speed dramatically you'll need to have the template read upfront when configuring XMLMill. Check the [XMLMill Configuration](#) chapter for more information.

4.1.3.2. Define the content of the acrofields.

In order to define the content of the template's acrofields two new elements have been defined:

4.1.3.2.1. Element: `ml:acroform`

The `ml:acroform` element just acts as a container element for the `ml:acrofield` elements.

```
<ml:acroform>
  <ml:acrofield .../>
  ...
</ml:acroform>
```

4.1.3.2.2. Element: `ml:acrofield`

The `ml:acrofield` element defines the content of a field contained in the PDF 'template' document.

```
<ml:acroform>
  <ml:acrofield name='InvoiceNumber'><xsl:value-of select='data/invoicenumber'
readonly='on' /> </ml:acrofield>
  <ml:acrofield name='InvoiceDate'><xsl:value-of select='data/invoicedate'
readonly='on' /></ml:acrofield>
  ...
</ml:acroform>
```

The **name** attribute defines the field to be filled (this is the field's name as defined in the PDF 'template' document).

The **readonly** defines if the field must be marked as readonly in the document to be generated.

4.1.3.2.3. Generate the document

When the document is generated following is done:

1. The 'template' document is parsed and so XMLMill knows which Acrofields exists in the template.
2. A line is written in the log file showing the names of the Acrofields detected.
3. The original template document is written in a new document and the fields are updated with the values as defined in the .xsl document using the 'incremental update' technology.

Below you'll find an example of the line in the log file:

```
DEBUG [main] Start reading template file: c:/templates/FormFieldsEnumerator.pdf
DEBUG [main] Acrofields: Button, CheckBox1, CheckBox2, TextBox1, TextBox2,
DEBUG [main] End reading template file: (491 elapsed (milliseconds))
```

☞ *Be sure that the loglevel is set to **DEBUG**.*

During the parsing of the template some errors can occur:

4.2. Stamp the page(s) of an existing PDF document

As of this version the regular `ml:` elements can be used to 'stamp' an existing PDF document. This way new information can be added on a document existing pages.

☞ *It is not possible with this version to add new pages to an existing document.*

The .xsl/.mill document should refer to an existing PDF document.

☞ *For a complete example please check the **Element_acrofield_FormFieldsEnumerator-stamp-only.mill** example in the **samples/mill/** directory contained in the download.*

4.2.1. Refer to an existing PDF document

In order to use the pdf document as a template following needs to be done:

1. Define the location of the PDF document acting as 'template'.
2. Use the regular `ml:` elements to 'stamp' the page(s).

4.2.1.1. Define the location of the 'template'.

The **ml:document** element has been extended with 2 new attributes:

4.2.1.1.1. Attribute: **template**

Value: <name>

Initial: an empty name

Applies to: ml:document

Inherited: No, a value is required

Description:

The **template** attribute defines the location of the PDF document that will act as 'template'.

Values have the following meaning:

<name> This value indicates the name (and location) of the PDF document (acting as 'template') to use.

Example:

```
<ml:document template="file:pdftemplates/FormFieldsEnumerator-final-1.3.pdf"
template-in-memory="on" ...>
```

4.2.1.1.2. Attribute: **template-in-memory**

Value: on | off | true | false

Initial: off

Applies to: ml:document

Inherited: No, a value is required

Description:

The **template-in-memory** attribute defines whether the template, after it has been parsed, must be kept in memory so a second reading of the document can be avoided at the moment of generating the new document based on this template.

```
<ml:document template="file:pdftemplates/FormFieldsEnumerator-final-1.3.pdf"
template-in-memory="on" ...>
```

- ☞ *This will not prevent that generating multiple documents with the same .xsl document the 'template' PDF document will each time be parsed again!*
- ☞ *To increase speed dramatically you'll need to have the template read upfront when configuring XMLMill. Check the [XMLMill Configuration](#) chapter for more information.*

4.3. Complete Acrofields and Stamp the page(s) of an existing PDF document

XMLMill can also fill the acrofields of a PDF document and at the same time stamp its' pages.

To do this the **ml:acroform** and other **ml:** elements are combined in one .xsl/.mill document

For more information please check the previous two sections [Complete Acrofields of an existing PDF document](#) and [Stamp the page\(s\) of an existing PDF document](#).

- ☞ *For a complete example please check the **Element_acrofield_FormFieldsEnumerator-form-and-stamp.mill** example in the **samples/mill/** directory contained in the download.*

4.4. XMLMill InterActive: New user-interface

The user-interface has been greatly enhanced, facilitating generating PDF documents interactively.

Following are the main changes:

- ◆ A **Splash screen** is added that will be shown during the indexation of the disks.
- ◆ A **Logging** tab is added to provide real-time logging statements as they are generated
- ◆ A **Save...** button is added on the **Logging** tab in order to save the log to a file.
- ◆ A **progress bar** is added on the **Logging** tab.
- ◆ a **Cancel** button is added on the **Logging** tab to interrupt the generation.

☞ *The interruption can only be done between the generation of 2 processes.*

4.4.1. Splash screen



A Splash screen is added that is shown during the indexation of the disks.

After the indexation is done the Splash screen can be closed using the **Continue** button.

4.4.2. Main Window



The main window has 2 tabs: **Dashboard** and **Logging**

4.4.2.1. Dashboard

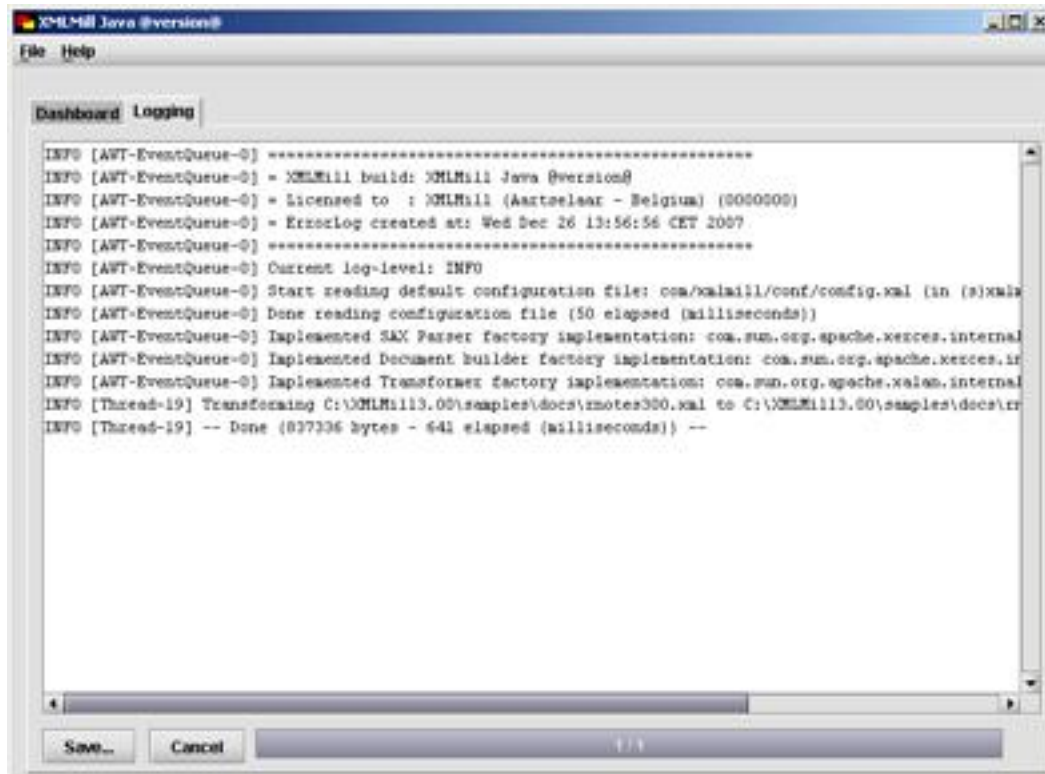
This tab contains several parts:

Parameters Contains the entries to define all details of the generation process

- Logging** Defines the level of logging.
- JAXP** Defines the JAXP version of the underlying transformer will XMLMill use.
- Abort On...** Defines when the generation process should be aborted.
- Validation** Defines if the intermediate `ml:` code should be validated.
- XSLT only** Defines that only the intermediate `ml:` code should be generated

☞ *For more information regarding above options please consult the [userguide document](#).*

4.4.2.2. Logging



During the generation process the log messages become visible in the logging tab.

This tab contains 2 buttons:

Save... Save the generated log messages to a file.

Cancel Aborts the generation process between generating 2 documents (in case you have selected multiple .xml/.mill documents in the previous tab).

The progress bar shows the number of documents generated (in case you have selected multiple .xml/.mill document in the previous tab).

4.5. XMLMill Logging

Uptill the previous version of XMLMill the `log4j` package was used to log the information generated during the generation process.

As this was sometimes a constraint for using XMLMill in some environments, the current version uses Apache's `commons-logging-1.1` package. This is an ultra-thin bridge between different logging implementations. This facilitates using XMLMill with any logging implementation at runtime.

☞ *Please visit the <http://jakarta.apache.org/commons/logging/> website for more information regarding the commons logging implementation.*

4.5.1. Class-path changes

Following needs to be done:

- ◆ Be sure Apache's `commons-logging-1.1.jar` (or higher) is available on the classpath.

- ◆ Be sure the underlying logging implementation is also available on the classpath (e.g. **log4j-*.jar**).

☞ *In case it is preferred to use the default JDK 1.4 (or higher) (**java.util.logging**.) logging mechanism the **log4j-*.jar** file can be omitted.*

4.5.2. Changing .java code

In order to use this version of XMLMill the code that calls the logger instance needs to be changed. Up to previous versions this was the general code:

```
public static void main(String[] args) {
    JAXPTransformer transform;

    try {
        // Set Log-level (optional)
        PDXLogger.setLevel("INFO");

        // Set the logfile (optional)
        PDXLogger.setPDXLogFile("file:/C:/XMLMill/log/phonelist.log", false);

        // Define a configurator (optional)
        Configurator.configure();

        ...
    }
}
```

The code started with first defining the loglevel (using the **PDXLogger.setLevel()** method) and then defining the logfile (using the **PDXLogger.setPDXLogFile** method).

After having defined these log parameters the configuration is defined (using the **Configurator.configure** method).

As of this version you need to:

- ❶ Define the classes of the underlying logging implementation.
- ❷ Define the Apache's commons logging class acting as a wrapper class against the underlying log implementation.
- ❸ Instantiate the wrapper class.
- ❹ Define the log level.
- ❺ Pass the wrapper to the Configurator, so XMLMill can use it.

4.5.2.1. Example: Using the **com.xmlmill.log.FileLogger**

The following example shows how to use the build-in **com.xmlmill.log.FileLogger** class.

```
package com.xmlmilltest;

import com.xmlmill.*;
import com.xmlmill.config.Configurator;

// The FileLogger class needed ❶
import com.xmlmill.log.FileLogger;

// The Apache commons logging class needed ❷
// NOT NEEDED import org.apache.commons.logging.impl.Log4JLogger;

import java.io.*;

public class TestAppl extends Object {

    public static void main(String[] args) {
        JAXPTransformer transform;
        FileLogger logger = null;

        try {
            //////////////////////////////////////
            // DEFINE FIRST THE LOGGER INSTANCE TO USE           //
            // HERE DEFAULT com.xmlmill.log.FileLogger           //
            // (you can use any class which implements the       //
            // org.apache.commons.logging.Log interface)         //
            //////////////////////////////////////
        }
    }
}
```

```

// Use the default FileLogger class ❸
logger = new FileLogger("xmlmill");

// Set logfile
logger.setFileName("file:/C:/xmlmill3.00/testtransform.txt", false);

// Set the loglevel ❹
logger.setLevel("DEBUG");

// Pass the logger to the XMLMill Configurator, so XMLMill can use it.
Configurator.setLogger(logger); ❺

////////////////////////////////////
// INITIALIZE THE CONFIGURATOR      //
////////////////////////////////////

// Set Configurator
Configurator.configure();

////////////////////////////////////
// CONTINUE                          //
////////////////////////////////////

// Remaining code
...

} catch (Throwable e) {
System.out.println(e.toString());
} finally {
// Close the logger instance
logger.finalize();

// Null out instances
transform = null;
logger = null;
System.out.println("-- Done --");
}
}
}

```

- ☞ *In case you want to use your own logging implementation, you need to write your own wrapper class implementing the **org.apache.commons.logging.Log** class.*
- ☞ *In case no wrapper is passed to the Configurator before calling the **Configurator.configure()** method, all log messages are send to the **System.err** stream.*

4.5.2.2. Example: Using the log4j package: org.apache.log4j

The following example shows how to use the **log4j** logging mechanism.

```

package com.xmlmilltest;

import com.xmlmill.*;
import com.xmlmill.config.Configurator;

// The Log4J classes needed ❶
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.PatternLayout;
import org.apache.log4j.FileAppender;
import org.apache.log4j.ConsoleAppender;

// The Apache commons logging class needed ❷
import org.apache.commons.logging.impl.Log4JLogger;

import java.io.*;

public class TestAppl extends Object {

public static void main(String[] args) {
JAXPTransformer transform;

try {
////////////////////////////////////
// DEFINE FIRST THE LOGGER INSTANCE TO USE //

```

```

//          HERE LOG4J LOGGER          //
////////////////////////////////////

// Get a wrapper instance around a Log4J logger instance ❸
Log4JLogger wrapper = new Log4JLogger("xmlmill");

// Set the loglevel ❹
wrapper.getLogger().setLevel(Level.INFO);

// Define pattern/formatter
PatternLayout patternlayout = new PatternLayout("%-5p [%t]: %m%n");

// Set Appender
wrapper.getLogger().addAppender(new FileAppender(patternlayout, ("%h/mylog.txt",
append)));

// Pass the logger to the XMLMill Configurator, so XMLMill can use it.
Configurator.setLogger(wrapper); ❺

////////////////////////////////////
// INITIALIZE THE CONFIGURATOR      //
////////////////////////////////////

// Define a configurator (optional)
Configurator.configure();

// Remaining code
...

} catch (Throwable e) {
    System.out.println(e.toString());
} finally {
    transform = null;
    System.out.println("-- Done --");
}
}
}

```

- ☞ Apache's **commons.logging** implementation has some wrapper classes defined for the most used log implementations. Check the **org.apache.commons.logging.impl** package for more information.
- ☞ In case you want to use your own logging implementation, you need to write your own wrapper class implementing the **org.apache.commons.logging.Log** class.
- ☞ In case no wrapper is passed to the Configurator before calling the **Configurator.configure()** method, all log messages are send to the **System.err**. stream.

4.5.2.3. Example: Using the JDK 1.4 package: `java.util.logging`

The following example shows how to use the `java.util.logging` logging mechanism.

```

package com.xmlmilltest;

import com.xmlmill.*;
import com.xmlmill.config.Configurator;

// The JDK 1.4 Logging classes needed
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.logging.FileHandler;

// The Apache commons logging class needed
import org.apache.commons.logging.impl.Jdk14Logger;

import java.io.*;

public class TestAppl extends Object {

    public static void main(String[] args) {
        JAXPTransformer transform;

        try {
            //////////////////////////////////////
            // DEFINE FIRST THE LOGGER INSTANCE TO USE //
            //          HERE JDK 1.4 LOGGER          //

```

```

////////////////////////////////////
// Get a wrapper instance around a JDK1.4 logger instance
Jdk14Logger wrapper = new Jdk14Logger("xmlmill");

// Get the underlying JDK1.4 logger instance and set loglevel
wrapper.getLogger().setLevel(Level.ALL);

// Define a handler
FileHandler f2 = new FileHandler("c:\\xmlmill3.00\\testtransform.txt");
f2.setFormatter(new SimpleFormatter());
wrapper.getLogger().addHandler(f2);

// Pass the logger to the XMLMill Configurator, so XMLMill can use it.
Configurator.setLogger(wrapper);

////////////////////////////////////
// INITIALIZE THE CONFIGURATOR           //
////////////////////////////////////

// Define a configurator (optional)
Configurator.configure();

// Remaining code
...

} catch (Throwable e) {
    System.out.println(e.toString());
} finally {
    transform = null;
    System.out.println("-- Done --");
}
}
}

```

- ☞ *In case you want to use your own logging implementation, you need to write your own wrapper class implementing the **org.apache.commons.logging.Log** class.*
- ☞ *Check also XMLMill's **com.xmlmill.log** package containing 2 wrapper classes (the **FileLogger** and **SimpleLogger** classes).*
- ☞ *In case no wrapper is passed to the Configurator before calling the **Configurator.configure()** method, all log messages are send to the **System.err** stream.*

4.6. XMLMill Configuration

4.6.1. Introduction

XMLMill has its own configuration file. This file is mainly used for:

- ◆ Overriding default values for the **<ml:simple-page-master>** attributes (margins, page-size, ...).
- ◆ Defining the font specifications of TrueType and **external** Type1 fonts.
- ◆ Defining the location of the Glyphlist files.
- ◆ Defining the JAXP settings.
- ◆ Defining the Digital Signature settings.

As of this version two new settings are added:

- ◆ Define some 'legacy' settings, facilitating the upgrade to this version.
- ◆ Define which PDF documents should be parsed (and kept into memory) as 'template'.

More information in the sections below.

4.6.2. Element: <legacy>

The **legacy** element contains some children that defines if XMLMill should be backwards

compatible with previous versions, facilitating the upgrade to this version.

4.6.2.1. Element: `<external-graphic>`

This element defines whether the DPI must be taken into consideration when reading an image (if defined in the image itself).

Uptill versions prior to 3.0 XMLMill assumed the DPI of an image was always 96. As of version 3.00 the DPI, if defined in the image file, will be used.

Example:

```
<!-- Define legacy rules for backward compatibility -->
<legacy>
  <external-graphic version="3.0" readDPIfromImage="on" />
</legacy>
```

The attributes are:

- version** Defines as of which version of XMLMill this element will be taken into account. Leave it always at **3.0**
- readDPIfromImage** Defines whether or not to take into consideration the DPI setting of an image, if defined in the image file. Possible values are **on** or **off**.

4.6.3. Element: `<templates>`

The **templates** element defines which PDF documents should be parsed as templates during the configuration process.

This will allow high volume generation of PDF documents based on these 'templates', as these templates will only be parsed once and its' content will be kept into memory, avoiding reading the template from the persistent storage over and over again.

4.6.3.1. Element: `<template>`

This element defines the location of the PDF document acting as template and how to treat it.

Example:

```
<!-- Define legacy rules for backward compatibility -->
<templates>
  <template pdf-file="file:/usr/templates/FormFieldsEnumerator.pdf"
  load-on-startup="on" template-in-memory="on" />
</templates>
```

The attributes are:

- pdf-file** Defines the location of the PDF document acting as 'template'.
- load-on-startup** Defines whether or not to parse this template when loading the configurator. Keep this always **on** except if the document should not be parsed during configuration.
- template-in-memory** Defines whether or not the PDF document should be kept in memory. Keeping the document in memory will avoid reading the document from persistent storage each time a new document is generated based on this template. Keep the value **on** except if the document is large and you do not want to keep it into memory.

☞ *Multiple `<template>` elements can be defined.*

☞ *If **load-on-startup** is **off**, the template document will not be kept into memory, even if **template-in-memory** is **on**.*

4.7. Implementation: **break-before** = 'column' attribute/value

As of this version a break can be forced in a column area.

4.8. Implementation: **break-after** = 'column' attribute/value

As of this version a break can be forced in a column area.

4.9. Rotate an image in a table-cell

As of this version an image can be rotated in a table-cell.

Example:

```
<ml:table ...>
  <ml:table-body ...>
    <ml:table-row>
      <ml:table-cell reference-orientation="90" ... >
        <ml:external-graphic src="file:dpibs/300DPI.jpg" />
      </ml:table-cell>
    </ml:table-row>
  </ml:table-body>
</ml:table>
```

To rotate the image you use the **reference-orientation** attribute of the **ml:table-cell**.

☞ Check the **Example_external-graphic-2-v300.mill** document in the **samples/mill** directory in the download as an example.

4.10. Behavioral change: Empty ml:textbox or ml:table-cell

Already as of version 2.80 there was a behavioral change regarding the way an empty ml:textbox or **ml:table-cell** is generated.

☞ This is documented in this [release notes document](#) as this was not documented in the [release notes of version 2.80](#).

4.10.1. Before version 2.80

Before version 2.80 an implicit space character was added to an empty ml:textbox or **ml:table-cell**. The result was that a textbox or table-cell had always a minimum height as defined by the space character (and depending on the font used).

This means that following example generates a textbox with an implicate space character added to the content.:

```
<ml:textbox border="red solid 5pt" width="5cm" />
```

The result is:



4.10.2. As of version 2.80

As of version 2.80 no implicit space character is added if the **ml:textbox** or **ml:table-cell** are empty.

This means that following example generates a genuine empty textbox:

```
<ml:textbox border="red solid 5pt" width="5cm" />
```

The result is:



This might impact the layout of your generated PDF documents. If needed add a space character to the **ml:textbox** or **ml:table-cell** content.

☞ Please check the **Element_textbox-empty.mill** example in the **samples/mill** directory for an example.

4.11. Behavioral change: The calculation of the width or height of an image

has changed

In previous versions the width or height of an image was wrongly calculated in case the image width or height was larger than its' parent content-area (the page or table-cell).

As of this version the calculation is more correctly and less confusing.

Assume following:

- ◆ An image with dimensions: **width=800px, height=60px**
- ◆ The image is printed on the page, the size of the page's content-area is: **width=470px, height=737px**
- ◆ The **ml:external-graphic** element has following attributes:
content-height="30px" content-width="auto" scaling="uniform"

Calculation up to version 2.80

1. The width of the image is lowered to 470px (maximum width of its' parent container (= the page area)).
2. The content-height is fixed to 30px.
3. The content-width is defined as follows: $30\text{px}/60\text{px} \times 470\text{px} = 235\text{px}$.
4. If the width is bigger than the parent's width (470px) it is lowered to 470px (not the case in this situation).

This led to a distorted view as the ratio is not maintained. The correct width should have been:
 $30\text{px}/60\text{px} \times 800\text{px} = 400\text{px}$.

Calculation as of version 3.00

1. The width of the image remains 800px (although it is larger than the maximum width of its' parent container (= the page area)).
2. The content-height is fixed to 30px.
3. The content-width is defined as follows: $30\text{px}/60\text{px} \times 800\text{px} = 400\text{px}$.
4. If the width is bigger than the parent's width (468px) it is lowered to 468px (not the case in this situation).

This width is correctly calculated, the aspect-ratio is maintained.

☞ *This change might impact the layout of your documents, as some images will no longer be distorted. Modify your code so your documents' layout remains the same.*

4.12. XMLMill Batch

Due to the no longer use of the **log4j** package following loglevels are suppressed:

- ◆ **ALL**
- ◆ **OFF**

Use the **TRACE** loglevel which replaces the **ALL** loglevel.

Do not mention the **--loglevel** parameter anymore in case you do not want to do any logging (instead of using the **OFF** parameter).

Do not mention the **--loglevel** parameter anymore in case you do not want to do any logging (instead of using the **OFF** parameter).

Please consult the [XMLMill Logging](#) section to find out how to set-up the logging behavior of XMLMill.

4.13. Removed classes

Due to the no longer use of the `log4j` package following deprecated classes were removed:

- ◆ `PDXLogger`
- ◆ `PDXLogFile`
- ◆ `PDELogFile`

Please consult the [XMLMill Logging](#) section to find out how to set-up the logging behavior of XMLMill.

Following class was also removed:

- ◆ `PDXTransform`

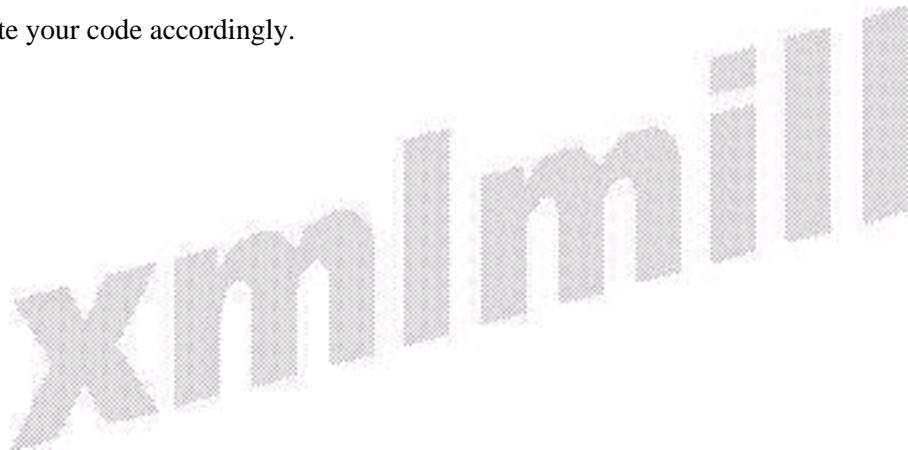
Please use the `JAXPTransformer` class instead.

4.14. Renamed classes

To make the API more logical and coherent, following class has been renamed:

- ◆ `PDXTransformErrorHandler` has become `JAXPErrorHandler`.

Please update your code accordingly.



5. XMLMill for Domino

5.1. Complete Acrofields of an existing PDF document

Please visit the [Complete Acrofields of an existing PDF document](#) chapter in this document.

5.2. Stamp the page(s) of an existing PDF document

Please visit the [Stamp the page\(s\) of an existing PDF document](#) chapter in this document.

5.3. Complete Acrofields and Stamp the page(s) of an existing PDF document

Please visit the [Complete Acrofields and Stamp the page\(s\) of an existing PDF document](#) chapter in this document.

5.4. XMLMill Configuration

Please visit the [XMLMill Configuration](#) chapter in this document.

5.5. Implementation: break-before = 'column' attribute/value

Please visit the [Implementation: break-before = 'column'](#) chapter in this document.

5.6. Implementation: break-after = 'column' attribute/value

Please visit the [Implementation: break-after = 'column'](#) chapter in this document.

5.7. Rotate an image in a table-cell

Please visit the [Rotate an image in a table-cell](#) chapter in this document.

5.8. Behavioral change: Empty ml:textbox or ml:table-cell

Please visit the [Behavioral change: Empty ml:textbox or ml:table-cell](#) chapter in this document.

5.9. Behavioral change: The calculation of the width or height of an image has changed

Please visit the [Behavioral change: The calculation of the width or height of an image has changed](#) chapter in this document.

6. Bugfixes

Following bugfixes were done:

- ◆ In te log file there was a reference to the .xsl file instead of the .mill/.xml file. This has been corrected.
- ◆ When a PDF document is encrypted an external reference (http, https, file,...) did not work. This has been corrected.
- ◆ The **com.xmlmill.connector.ant.Generator** class now writes a list of files to the log file that will be generated.
- ◆ The **com.xmlmill.connector.ant.Comparer** class now writes a list of master PDF files to the log file that are selected to be compared.
- ◆ A **java.lang.NullPointerException** could occur sometimes when an **XSLTTransformationOnly** needed to be done. This has been corrected.
- ◆ An error was detected in the **Fetch2.java** class in the servlets examples.

xmlmill

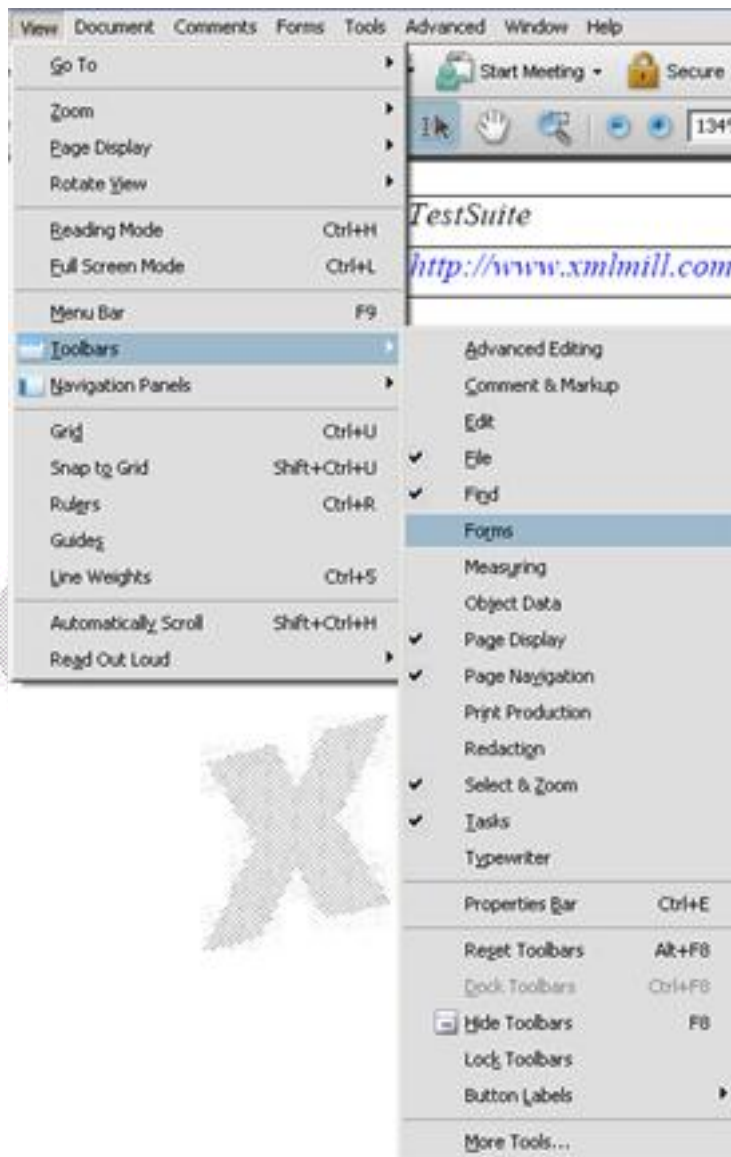
7. Appendix: Adding Acrofields to a PDF document with Acrobat Professional 8

7.1. Modify an existing document with Acrobat Professional (8.0)

*You cannot use Adobe LifeCycle Designer (ALD) to add acrofields on a form. The reason is that ALD - in some cases - makes from the fields fully qualified fields (like `topmostSubform[0].Page1[0].allbutlastname[0]`). This is not a problem for XMLMill but the problem is that Acrobat Reader shows an *empty* qualified field, even when it has been filled with XMLMill. Other Acrobat readers (like Foxit Reader) do show the filled qualified fields correctly.*

To add Acrofields to an existing document in Acrobat Professional (8.0) do as follows:

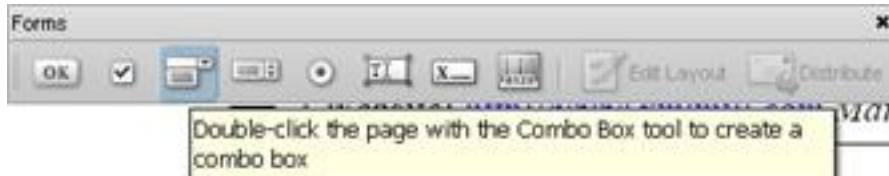
Activate the **Forms** toolbar:



The **Forms** toolbar will pop up.:



Start using the different form elements to define your field:



☞ Mark all the fields as **read-only** so users of generated documents based on this 'template' will not be able to change the contents of the acrofields.

Optimize the document:

Once all fields are created, optimize the PDF document:



Make it compatible with Acrobat 4 (preferred):

